RNAseq_DE

Matthew Taliaferro

6/16/2020

```
library(tximport)
library(DESeq2)
library(tidyverse)
library(knitr)
library(RColorBrewer)
library(ggrepel)
library(pheatmap)
library(biomaRt)
library(reshape2)
library(broom)
```

Overview

Last time we took RNAseq data from an *in vitro* differentiation timecouse from mouse ESCs to glutaminergic neurons (Hubbard et al, F1000 Research (2013)). We took in transcript-level quantifications produced by `salmon` and collapsed them to gene-level quantifications using `tximport`. We then inspected the quality of the data by relating distances between samples using two methods: hierarchical clustering and principal components analysis. We found that the data was of high quality, as evidenced by the fact that replicates from a given timepoint were highly similar to other replicates for the same timepoint, and the distances between samples made sense with what we know about how the experiment was conducted.

Today, we are going to pretend that this isn't a timecourse. We are going to imagine that we have only two conditions: the first timepoint (DIVminus8) and the last timepoint (DIV28). We will use `DESeq2` to identify genes that are differentially expressed between these two timepoints. We will then plot changes in expression for both individual genes and groups of genes that we already know going in might be interesting to look at. Finally, we will look at some features of transcripts and genes that are differentially expressed between these two timepoints. We are not doing this two-timepoint comparsion because it is necessarily a good thing to do with timepoint data. Instead, we are doing this because often in an RNAseq experiment, you only have two conditions to compare.

Identifying differentially expressed genes with DESeq2

The first thing we need to do is read in the data again and move from transcript-level expression values to gene-level expression values with `tximport`. Let's use `biomaRt` to get a table that relates gene and transcript IDs.

```
mart <- biomaRt::useMart("ENSEMBL_MART_ENSEMBL", dataset = "mmusculus_gene_ensembl", host='uswest.ensembl
t2g <- biomaRt::getBM(attributes = c('ensembl_transcript_id', 'ensembl_gene_id', 'external_gene_name'),
```

```
## Warning: `select_()` is deprecated as of dplyr 0.7.0.
## Please use `select()` instead.
## This warning is displayed once every 8 hours.
```

```
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning: `filter_()` is deprecated as of dplyr 0.7.0.
## Please use `filter()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Cache found
```

Now we can read in the transcript-level data and collapse to gene-level data with `tximport`

```
#The directory where all of the sample-specific salmon subdirectories live
base_dir <- '../data/salmonouts/'

#The names of all the sample-specific salmon subdirectories
sample_ids <- c('DIV0.Rep1', 'DIV0.Rep2', 'DIV0.Rep3',
                'DIV7.Rep1', 'DIV7.Rep2', 'DIV7.Rep3', 'DIV7.Rep4')

#So what we want to do now is create paths to each quant.sf file that is in each sample_id.
#This can be done by combining the base_dir, each sample_id directory, and 'quant.sf'
#For example, the path to the first file will be data/salmonouts/DIVminus8.Rep1/quant.sf
salm_dirs <- sapply(sample_ids, function(id) file.path(base_dir, id, 'quant.sf'))

#Run tximport
txi <- tximport(salm_dirs, type = 'salmon', tx2gene = t2g, dropInfReps = TRUE, countsFromAbundance = 'l
```

```
## reading in files with read_tsv
```

```
## 1 2 3 4 5 6 7
## transcripts missing from tx2gene: 602
## summarizing abundance
## summarizing counts
## summarizing length
```

OK we are going to need to give `DESeq2` information about the samples like which sample belongs to which timepoint.

```
samples <- data.frame(row.names = c('DIV0.Rep1', 'DIV0.Rep2', 'DIV0.Rep3',
                                    'DIV7.Rep1', 'DIV7.Rep2', 'DIV7.Rep3', 'DIV7.Rep4'),
                      timepoint = c(rep('DIV0', 3), rep('DIV7', 4)))

head(samples)
```

```
##           timepoint
## DIV0.Rep1      DIV0
## DIV0.Rep2      DIV0
## DIV0.Rep3      DIV0
## DIV7.Rep1      DIV7
## DIV7.Rep2      DIV7
## DIV7.Rep3      DIV7
```

There are essentially two steps to using `DESeq2`. The first involves creating a `DESeqDataSet` from your data. Luckily, if you have a `tximport` object, which we do in the form of `txi`, then this becomes easy.

```
ddsTxi <- DESeqDataSetFromTximport(txi, colData = samples, design = ~ timepoint)
```

```
## using just counts from tximport
```

You can see that `DESeqDataSetFromTximport` wants three things. The first is our `tximport` object. The second is the dataframe we made that relates samples and conditions (or in this case timepoints). The last is something called a **design formula**. A design formula contains all of the variables that will go into `DESeq2`'s model. The formula starts with a tilde and then has variables separated by a plus sign. It is common practice, and in fact basically required with `DESeq2`, to put the variable of interest last. In our case, that's trivial because we only have one: timepoint. So our design formula is very simple:

```
design = ~ timepoint
```

Your design formula should ideally include **all of the sources of variation in your data**. For example, let's say that here we thought there was a batch effect with the replicates. Maybe all of the Rep1 samples were prepped and sequenced on a different day than the Rep2 samples and so on. We could potentially account for this in `DESeq2`'s model with the following forumula:

```
design = ~ replicate + timepoint
```

Here, timepoint is still the variable of interest, but we are controlling for differences that arise due to differences in replicates. Of course, this formula would require us to go back and make a new `samples` table that included 'replicate' as a factor.

```
ddsTxi <- DESeqDataSetFromTximport(txi, colData = samples, design = ~ timepoint)
```

```
## using just counts from tximport
```

```
ddsTxi
```

```
## class: DESeqDataSet
## dim: 52346 7
## metadata(1): version
## assays(1): counts
## rownames(52346): ENSMUSG00000000001 ENSMUSG00000000003 ...
##   ENSMUSG00000118655 ENSMUSG00000118659
## rowData names(0):
## colnames(7): DIV0.Rep1 DIV0.Rep2 ... DIV7.Rep3 DIV7.Rep4
## colData names(1): timepoint
```

We can see here that `DESeq2` is taking the counts produced by `tximport` for gene quantifications. There are 52346 genes (rows) here and 7 samples (columns).

Now using this ddsTxi object, we can run `DESeq2`.

```
dds <- DESeq(ddsTxi)
```

```
## estimating size factors
```

```
## estimating dispersions
```

```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
```

```
## fitting model and testing
```

There are many useful things in this `dds` object. I encourage you to take a look at the vignette for `DESeq2` for a full explanation about what is in there, as well as info on many more tests and analyses that can be done with `DESeq2`.

The results can be accessed using the `results()` function. We are going to include the `contrast` argument here. `DESeq2` reports changes in RNA abundance between two samples as a log2FoldChange. That's great, but it's often not clear exactly what the numerator and denominator of that fold change ratio is. In this case, it could be either DIV7/DIV0 or DIV0/DIV7.

In actuality, it's of course not random. The alphabetically first condition will be the numerator. Still, I find it less confusing to explicitly specify what the numerator and denominator of this ratio are using the `contrast` argument.

Another useful application of the `contrast` argument can be seen with more complicated design formulae. Remember our design formula that accounted for potential differences due to Replicate batch effects:

```
~ replicate + timepoint
```

As we said before, with this formula, `DESeq2` will account for differences between replicates here to find differences between timepoints. However, what if we wanted to look at differences between replicate batches? By supplying 'replicate' to `contrast` instead of 'timepoint', we could extract difference between these batches.

```
#For contrast, we give three strings: the factor we are interested in, the numerator, and the denominat
#It makes the most sense (to me at least) to have DIV28 be the numerator
results(dds, contrast = c('timepoint', 'DIV7', 'DIV0'))
```

```
## log2 fold change (MLE): timepoint DIV7 vs DIV0
## Wald test p-value: timepoint DIV7 vs DIV0
## DataFrame with 52346 rows and 6 columns
##                            baseMean     log2FoldChange             lfcSE
##                           <numeric>          <numeric>         <numeric>
## ENSMUSG00000000001 5579.77754585436 -0.953512002710987 0.0527959783401513
## ENSMUSG00000000003                0                 NA                NA
## ENSMUSG00000000028 718.696881667605  -2.55365044138704  0.107061390183316
## ENSMUSG00000000031 3624.57797113539   4.12075207975084  0.289850192996735
## ENSMUSG00000000037 242.620316811254  -1.27535503985687  0.199245035001358
## ...                             ...                ...               ...
## ENSMUSG00000118642 327.352114985249  -3.25323665588743   0.23412080090127
## ENSMUSG00000118643 0.41966839171392  -1.01604388048036   2.91302113145088
## ENSMUSG00000118645                0                 NA                NA
## ENSMUSG00000118655 0.53983052909861    -0.203467630204   3.47258609246081
## ENSMUSG00000118659                0                 NA                NA
##                                 stat               pvalue
##                            <numeric>            <numeric>
## ENSMUSG00000000001   -18.0603150597522  6.54479924381713e-73
## ENSMUSG00000000003                  NA                   NA
## ENSMUSG00000000028    -23.852207009591 9.60625035722693e-126
## ENSMUSG00000000031    14.2168340036167  7.20390891834315e-46
## ENSMUSG00000000037   -6.40093761858697  1.54425661878405e-10
## ...                             ...                  ...
## ENSMUSG00000118642   -13.8955472703143   6.7406320059608e-44
## ENSMUSG00000118643  -0.348793858551345      0.72724407438924
## ENSMUSG00000118645                  NA                   NA
## ENSMUSG00000118655 -0.0585925373155588     0.953276654831871
## ENSMUSG00000118659                  NA                   NA
##                                  padj
##                             <numeric>
## ENSMUSG00000000001  8.45277461659094e-72
## ENSMUSG00000000003                    NA
## ENSMUSG00000000028 2.24778563098477e-124
## ENSMUSG00000000031  6.26890306369821e-45
## ENSMUSG00000000037   6.1371069162833e-10
## ...                                  ...
## ENSMUSG00000118642  5.70293586948921e-43
## ENSMUSG00000118643     0.907559578565775
```

```
## ENSMUSG00000118645                      NA
## ENSMUSG00000118655     0.976598345033357
## ENSMUSG00000118659                      NA
```

We can see here that this is a dataframe where the rows are genes and the columns are interesting data. The columns we are most interested in are **log2FoldChange** and **padj**. log2FoldChange is self-explanatory. padj is the pvalue for a test asking if the expression of this gene is different between the two conditions. This pvalue has been corrected for multiple hypothesis testing using the Benjamini-Hochberg method.

Let's do a little work on this dataframe to make it slightly cleaner and more informative.

```
differentiation.results <- results(dds, contrast = c('timepoint', 'DIV7', 'DIV0')) %>%
  #Change this into a dataframe
  as.data.frame(.) %>%
  #Move ensembl gene IDs into their own column
  rownames_to_column(., var = 'ensembl_gene_id') %>%
  #Get rid of columns that are so useful to us right now
  dplyr::select(., -c(baseMean, lfcSE, stat, pvalue)) %>%
  #Merge this with a table relating ensembl_gene_id with gene short names
  inner_join(unique(dplyr::select(t2g, -ensembl_transcript_id)), ., by = 'ensembl_gene_id') %>%
  #Rename external_gene_name column
  dplyr::rename(., Gene = external_gene_name)
```

OK now we have a table of gene expression results. How many genes are significantly up/down regulated between these two timepoints? We will use 0.01 as an FDR (p.adj) cutoff.

```
#number of upregulated genes
nrow(filter(differentiation.results, padj < 0.01 & log2FoldChange > 0))
```

```
## [1] 6671
```

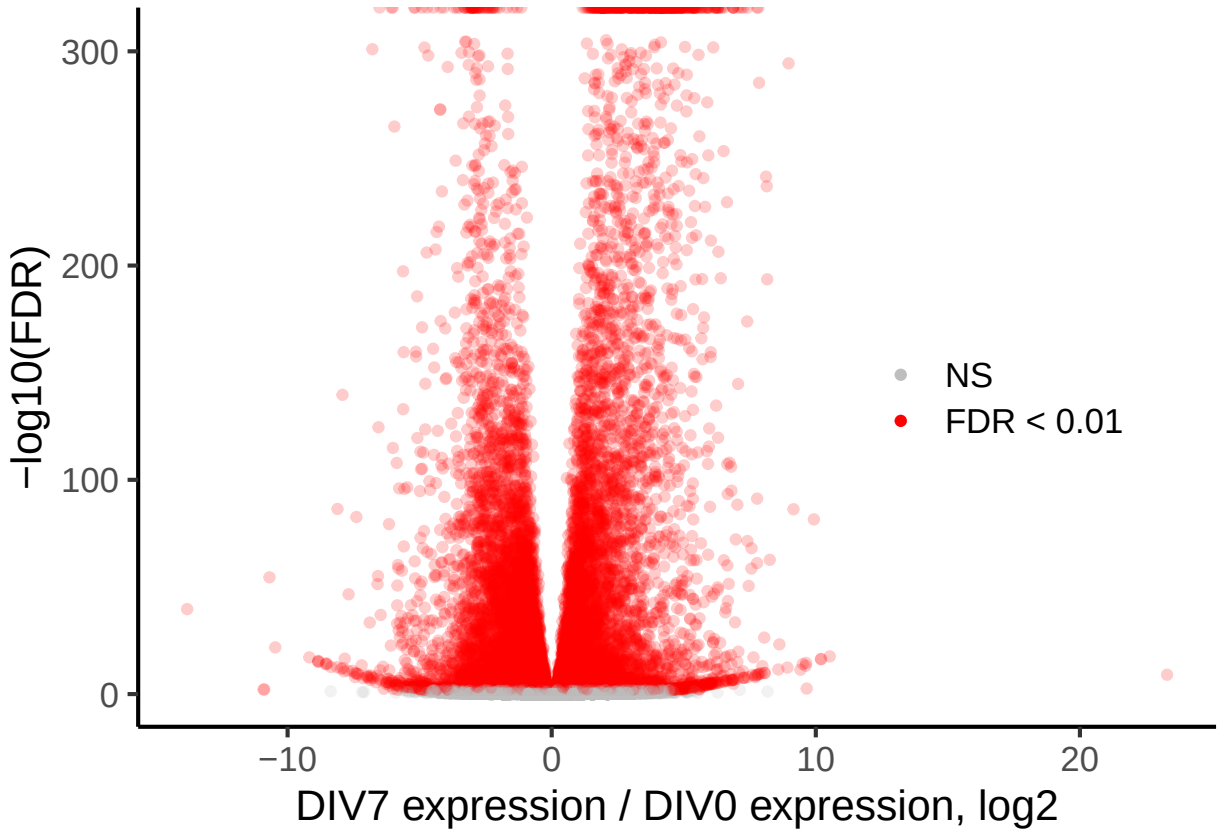
```
#number of downregulated genes
nrow(filter(differentiation.results, padj < 0.01 & log2FoldChange < 0))
```

```
## [1] 6813
```

Volcano plot of differential expression results

Let's make a volcano plot of these results.

```
#We are going to add an extra column to differentiation.results that tells us whether or not a gene
#meets the FDR cutoff

differentiation.results.sig <- mutate(differentiation.results, sig = ifelse(padj < 0.01, 'yes', 'no')) %
  #if a gene did not meet expression cutoffs that DESeq2 automatically does, it gets a pvalue of NA
  na.omit(.)

ggplot(differentiation.results.sig, aes(x = log2FoldChange, y = -log10(padj), color = sig)) +
  geom_point(alpha = 0.2) + xlab('DIV7 expression / DIV0 expression, log2') +
  ylab('-log10(FDR)') + theme_classic(16) +
  scale_color_manual(values = c('gray', 'red'), labels = c('NS', 'FDR < 0.01'), name = '') +
  theme(legend.position = c(0.8, 0.5)) + guides(color = guide_legend(override.aes = list(alpha = 1)))
```



OK that's a lot of significant genes. What if, in addition to an FDR cutoff, we applied a log2FoldChange cutoff? We can do this by asking for p values that incorporate the probability that the log2FoldChange was greater than a threshold. This will of course be more conservative, but will probably give you a more confident set of genes.

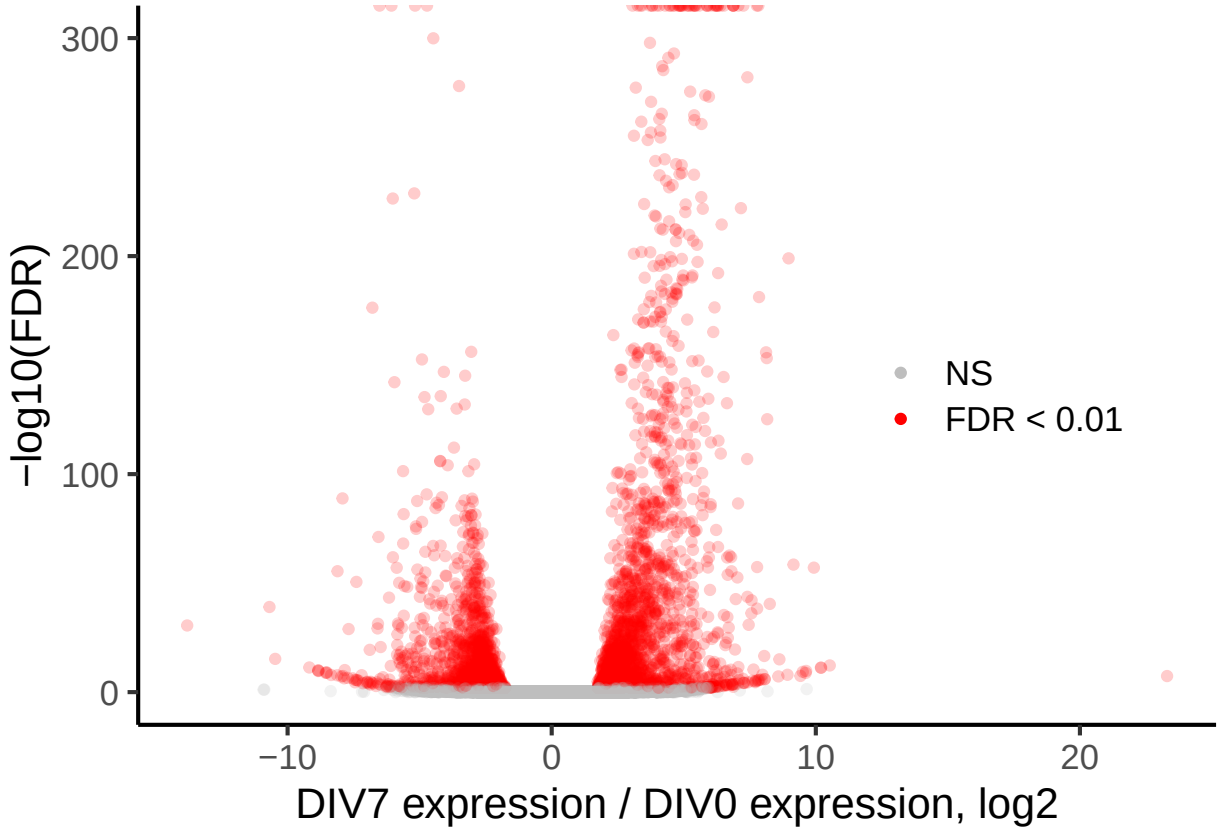
```
#Is the expression of the gene at least 3-fold different?
differentiation.results.lfc <- results(dds, contrast = c('timepoint', 'DIV7', 'DIV0'), lfcThreshold = l
  #Change this into a dataframe
  as.data.frame(.) %>%
  #Move ensembl gene IDs into their own column
  rownames_to_column(., var = 'ensembl_gene_id') %>%
  #Get rid of columns that are so useful to us right now
  dplyr::select(., -c(baseMean, lfcSE, stat, pvalue)) %>%
  #Merge this with a table relating ensembl_gene_id with gene short names
  inner_join(unique(dplyr::select(t2g, -ensembl_transcript_id)), ., by = 'ensembl_gene_id') %>%
  #Rename external_gene_name column
  dplyr::rename(., Gene = external_gene_name) %>%
  mutate(., sig = ifelse(padj < 0.01, 'yes', 'no')) %>%
  na.omit(.)

#number of upregulated genes
nrow(filter(differentiation.results.lfc, padj < 0.01 & log2FoldChange > 0))
```

```
## [1] 1834
```

```
#number of downregulated genes
nrow(filter(differentiation.results.lfc, padj < 0.01 & log2FoldChange < 0))
```

```
## [1] 1295
```

```
ggplot(differentiation.results.lfc, aes(x = log2FoldChange, y = -log10(padj), color = sig)) +
  geom_point(alpha = 0.2) + xlab('DIV7 expression / DIV0 expression, log2') +
  ylab('-log10(FDR)') + theme_classic(16) +
  scale_color_manual(values = c('gray', 'red'), labels = c('NS', 'FDR < 0.01'), name = '') +
  theme(legend.position = c(0.8, 0.5)) + guides(color = guide_legend(override.aes = list(alpha = 1)))
```



Plotting the expression of single genes

Sometimes we will have particular marker genes that we might want to highlight to give confidence that the experiment worked as expected. We can plot the expression of these genes in each replicate. For this case, let's plot the expression of two pluripotency genes (which we expect to decrease) and two strongly neuronal genes (which we expect to increase).

So what is the value that we would plot? Probably the most correct value is the 'normalized counts' value provided by `DESeq2`. This value is the safest to compare across samples within a gene. These counts are raw counts that have been normalized for sequencing depth and sample composition. However, I find that it is difficult to quickly get a sense of the expression level of a gene from it's number of normalized counts. Let's say a gene had 500 normalized counts. Is that a highly expressed gene? A lowly expressed gene? Well, I won't really know unless I knew the length of the gene, and I don't have the length of every gene memorized.

A more interpretable value to plot might be TPM, since TPM is length-normalized. Let's say a gene was expressed at 500 TPM. Right off the bat, I know generally what kind of expression that reflects (pretty high).

```
#If you are interested in getting normalized counts, here's how you do it
normcounts <- counts(dds, normalized = TRUE)
```

Let's plot the expression of Klf4, Sox2, Bdnf, and Dlg4 in our samples.

```
#Get a table of tpms...remember txi is our tximport object
tpms <- txi$abundance %>%
  as.data.frame(.) %>%
  rownames_to_column(., var = 'ensembl_gene_id') %>%
  inner_join(unique(dplyr::select(t2g, -ensembl_transcript_id)), ., by = 'ensembl_gene_id') %>%
  dplyr::rename(., Gene = external_gene_name) %>%
  #Filter for genes we are interested in
  filter(., Gene %in% c('Klf4', 'Sox2', 'Bdnf', 'Dlg4'))

tpms
```

```
##      ensembl_gene_id Gene  DIV0.Rep1  DIV0.Rep2  DIV0.Rep3  DIV7.Rep1
## 1 ENSMUSG00000003032 Klf4  40.022629  46.241797  57.374566   9.063035
## 2 ENSMUSG00000048482 Bdnf   2.456452   2.401157   2.351342   7.799981
## 3 ENSMUSG00000020886 Dlg4  18.752921  15.749749  12.212636 151.558403
## 4 ENSMUSG00000074637 Sox2 131.032132 120.433905 110.664468  24.484942
##    DIV7.Rep2  DIV7.Rep3  DIV7.Rep4
## 1   9.208364   9.715119   8.817037
## 2   7.762255   7.638515   7.433529
## 3 156.043098 153.269475 153.538436
## 4  26.080394  27.106054  26.995461
```

OK, this is close to what we want, but in order to plot it we need to turn it from a wide table into a long table.

```
tpms <- reshape2::melt(tpms, id.vars = c('ensembl_gene_id', 'Gene'))
head(tpms)
```

```
##      ensembl_gene_id Gene  variable      value
## 1 ENSMUSG00000003032 Klf4 DIV0.Rep1  40.022629
## 2 ENSMUSG00000048482 Bdnf DIV0.Rep1   2.456452
## 3 ENSMUSG00000020886 Dlg4 DIV0.Rep1  18.752921
## 4 ENSMUSG00000074637 Sox2 DIV0.Rep1 131.032132
## 5 ENSMUSG00000003032 Klf4 DIV0.Rep2  46.241797
## 6 ENSMUSG00000048482 Bdnf DIV0.Rep2   2.401157
```

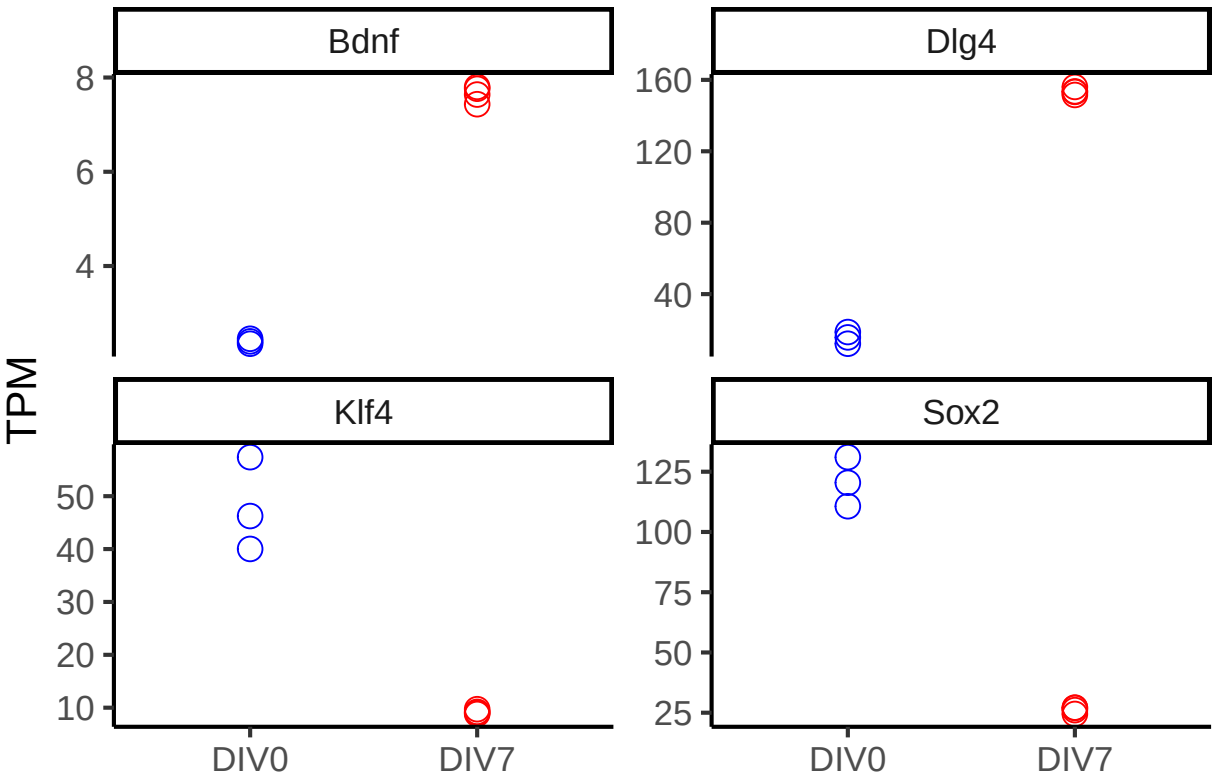
Now add one more column that tells which condition each replicate belongs to.

```
tpms <- mutate(tpms, condition = ifelse(grepl('DIV0', variable), 'DIV0', 'DIV7'))
head(tpms)
```

```
##      ensembl_gene_id Gene  variable      value condition
## 1 ENSMUSG00000003032 Klf4 DIV0.Rep1  40.022629      DIV0
## 2 ENSMUSG00000048482 Bdnf DIV0.Rep1   2.456452      DIV0
## 3 ENSMUSG00000020886 Dlg4 DIV0.Rep1  18.752921      DIV0
## 4 ENSMUSG00000074637 Sox2 DIV0.Rep1 131.032132      DIV0
## 5 ENSMUSG00000003032 Klf4 DIV0.Rep2  46.241797      DIV0
## 6 ENSMUSG00000048482 Bdnf DIV0.Rep2   2.401157      DIV0
```

Now plot.

```
ggplot(tpms, aes(x = condition, y = value, color = condition)) + geom_point(pch = 21, size = 4) +
  ylab('TPM') + xlab('') + theme_classic(16) + scale_color_manual(values = c('blue', 'red'), guide = F) +
  facet_wrap(~Gene, scales = 'free_y')
```



OK, but let's do something a little bigger. Say that instead of plotting individual genes we wanted to ask whether a whole class of genes are going up or down. We can do that by retrieving all genes that belong to a particular gene ontology term.

There are three classes of genes we will look at here: - Maintenance of pluripotency (GO:0019827) - Positive regulation of the cell cycle (GO:0045787) - Neuronal differentitaion (GO:0030182)

We can use `biomaRt` to get all genes that belong to each of these categories. Think of it like doing a gene ontology enrichment analysis in reverse.

```
pluripotencygenes <- getBM(attributes = c('ensembl_gene_id'), filters = c('go_parent_term'),
                           values = c('GO:0019827'), mart = mart)
```

```
## Cache found
```

```
cellcyclegenes <- getBM(attributes = c('ensembl_gene_id'), filters = c('go_parent_term'),
                           values = c('GO:0045787'), mart = mart)
```

```
## Cache found
```

```
neurongenes <- getBM(attributes = c('ensembl_gene_id'), filters = c('go_parent_term'),
                           values = c('GO:0030182'), mart = mart)
```

```
## Cache found
```

```
head(pluripotencygenes)
```

```
##      ensembl_gene_id
## 1 ENSMUSG00000028640
## 2 ENSMUSG00000027612
## 3 ENSMUSG00000063382
## 4 ENSMUSG00000008999
## 5 ENSMUSG00000042275
```

```
## 6 ENSMUSG00000050966
```

You can see that these items are one-column dataframes that have the column name 'ensembl_gene_id'. We can now go through our results dataframe and add an annotation column that marks whether the gene is in any of these categories.

```
differentiation.results.annot <- differentiation.results %>%
  mutate(., annot = case_when(ensembl_gene_id %in% pluripotencygenes$ensembl_gene_id ~ 'pluripotency',
                       ensembl_gene_id %in% cellcyclegenes$ensembl_gene_id ~ 'cellcycle',
                       ensembl_gene_id %in% neurongenes$ensembl_gene_id ~ 'neurondiff',
                       TRUE ~ 'none'))

#Reorder these for plotting purposes
differentiation.results.annot$annot <- factor(differentiation.results.annot$annot,
                                              levels = c('none', 'cellcycle', 'pluripotency', 'neurondi

head(differentiation.results.annot)
```

```
##      ensembl_gene_id    Gene log2FoldChange          padj annot
## 1 ENSMUSG00000064372   mt-Tp     2.98546451 3.529720e-123  none
## 2 ENSMUSG00000064371   mt-Tt    -2.96476297  5.664116e-03  none
## 3 ENSMUSG00000064370 mt-Cytb     0.25211246  2.560805e-03  none
## 4 ENSMUSG00000064369   mt-Te    -0.07304612  9.203204e-01  none
## 5 ENSMUSG00000064368  mt-Nd6     0.29691299  1.109676e-03  none
## 6 ENSMUSG00000064367  mt-Nd5     0.20923893  1.635304e-02  none
```

OK we've got our table, now we are going to ask if the log2FoldChange values for the genes in each of these classes are different that what we would expect. So what is the expected value? Well, we have a distribution of log2 fold changes for all the genes that are **not** in any of these categories. So we will ask if the distribution of log2 fold changes for each gene category is different than that null distribution.

You can tie yourself in knots worrying about what is the right test to use for a lot of these things. What are the assumptions of the test? Does my data fit these assumptions? Is it normally distributed? Blah blah blah. My advice is to pretty much always just use rank-based nonparametric tests. Yes, you will sacrifice some power, meaning that your p values will generally be higher than those produced by parametric tests that make assumptions. However, in genomics, we often have many observations of something or many members in our groups. So if your p value is borderline where the choice of test makes a big difference, I'm already a little skeptical.

```
p.pluripotency <- tidy(wilcox.test(filter(differentiation.results.annot, annot == 'pluripotency')$log2F
                           filter(differentiation.results.annot, annot == 'none')$log2FoldChange))

p.pluripotency <- p.pluripotency$p.value
p.pluripotency <- signif(p.pluripotency, 2)

p.cellcycle <- tidy(wilcox.test(filter(differentiation.results.annot, annot == 'cellcycle')$log2FoldChar
                           filter(differentiation.results.annot, annot == 'none')$log2FoldChange))

p.cellcycle <- p.cellcycle$p.value
p.cellcycle <- signif(p.cellcycle, 2)

p.neurondiff <- tidy(wilcox.test(filter(differentiation.results.annot, annot == 'neurondiff')$log2FoldCh
                           filter(differentiation.results.annot, annot == 'none')$log2FoldChange))

p.neurondiff <- p.neurondiff$p.value
p.neurondiff <- signif(p.neurondiff, 2)
```

OK now lets plot.

```
ggplot(differentiation.results.annot, aes(x = annot, y = log2FoldChange, color = annot)) +
  xlab('Gene class') + ylab('DIV7 expression / DIV0 expression, log2') +
  geom_hline(yintercept = 0, color = 'gray', linetype = 'dashed') +
  geom_boxplot(notch = TRUE, outlier.shape = NA) +
  theme_classic(14) + scale_color_manual(values = c('gray', 'red', 'blue', 'purple'), guide = F) +
  scale_x_discrete(labels = c('none', 'Cell cycle', 'Pluripotency', 'Neuron\ndifferentiation')) +
  coord_cartesian(ylim = c(-5, 8)) +
  annotate('segment', x = 1, xend = 2, y = 4, yend = 4) +
  annotate('segment', x = 1, xend = 3, y = 5, yend = 5) +
  annotate('segment', x = 1, xend = 4, y = 6, yend = 6) +
  annotate('text', x = 1.5, y = 4.4, label = paste0('p = ', p.cellcycle)) +
  annotate('text', x = 2, y = 5.4, label = paste0('p = ', p.pluripotency)) +
  annotate('text', x = 2.5, y = 6.4, label = paste0('p = ', p.neurondiff))
```

```
## Warning: Removed 17282 rows containing non-finite values (stat_boxplot).
```